

Instantons causing iterative decoding to cycle

Misha Stepanov

Abstract—It is speculated that the most probable channel noise realizations (instantons) that cause the iterative decoding of low-density parity-check codes to fail make the decoding not to converge. A simple example is given of an instanton that is not a pseudo-codeword and causes iterative decoding to cycle. A method of finding the instantons for large number of iterations is presented and tested on Tanner’s [155, 64, 20] code and Gaussian channel. The inherently dynamic instanton with effective distance of 11.475333 is found.

Index Terms—iterative decoding, LDPC codes, error floor.

I. INTRODUCTION

Low-density parity-check (LDPC) codes [1], [2], [3] with iterative decoding got a lot of attention due to their excellent performance. The decoding error probability is larger than one could expect when the Signal-to-Noise Ratio (SNR) is high, a phenomenon called error floor [4], [5].

In some cases the substructures of the code that provide a leading contribution to the error probability are known: they are *codewords* in the case of maximum likelihood decoding, and *stopping sets* [6] in the case of iterative decoding and binary erasure channel. For general situation several heuristics were introduced: *near-codewords* [4] or *trapping sets* [5] as bits subsets that violate just a few parity checks, *pseudo-codewords* [3] as the codewords on *computational tree*, *stopping sets*, *pseudo-codewords* as non-codeword vertices of a polytope used in linear programming decoding [7], *fully absorbing sets* [8], and *instantons* [9]. Even if the description of the deleterious substructures is available, it still could be a non-trivial problem to find them.

LDPC codes can be defined by parity check matrix \hat{H} or Tanner graph [10] which is a sparse bipartite graph with two sets of vertices: bits and parity checks. The notation $i \circ \square \alpha$ is used to indicate that $H_{\alpha i} = 1$ and the bit i and the check α are connected by an edge.

The binary (made of $+1$ and -1 numbers, or just “+”s and “−”s) codeword $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$ is transmitted over a noisy channel with continuous output $\mathbf{x} = (x_1, x_2, \dots, x_N)$. In the paper the channel is assumed to be memoryless, *i.e.*, $P(\mathbf{x}|\sigma) = \prod_{i=1}^N P(x_i|\sigma_i)$. The decoder takes the logarithmic likelihoods $h_i = (1/2) \log(P(+|x_i)/P(-|x_i))$ at each bit i as an input, where $P(\pm|x) = P(x|\pm)/(P(x|+) + P(x|-))$.

The iterative decoding that is used in the paper is the min-

sum algorithm

$$\begin{aligned} \text{decoding output: } m_i^{(k)} &= h_i + \sum_{\alpha}^{i \circ \square \alpha} \mu_{\alpha \rightarrow i}^{(k-1/2)} \\ \text{bits} \rightarrow \text{checks: } \eta_{i \rightarrow \alpha}^{(k)} &= h_i + \sum_{\beta \neq \alpha}^{i \circ \square \beta} \mu_{\beta \rightarrow i}^{(k-1/2)} \\ \text{messages} & \\ \text{checks} \rightarrow \text{bits: } \mu_{\alpha \rightarrow i}^{(k+1/2)} &= \min_{j \neq i}^{j \circ \square \alpha} |\eta_{j \rightarrow \alpha}^{(k)}| \cdot \prod_{j \neq i}^{j \circ \square \alpha} \text{sign} \eta_{j \rightarrow \alpha}^{(k)} \end{aligned}$$

with checking at each iteration whether the current output $\sigma = \text{sign} \mathbf{m}$ is a valid codeword, *i.e.*, $\hat{H} \cdot ((1 - \sigma)/2) = \mathbf{0} \pmod{2}$ (and if it is, the iterations stop). At the beginning of the decoding there are no messages to bits, $\mu_{\alpha \rightarrow i}^{(-1/2)} \equiv 0$.

Let us define the noise vector $\xi = (\xi_1, \xi_2, \dots, \xi_N)$ by $\xi_i = 1 - \sigma_i x_i$. For simplicity, the channel is assumed to be symmetric, $P(-x|+) = P(x|-)$, then the decoding error probability (and error causing noise configurations) is independent of the codeword σ being sent.

Consider the error correcting code, the transmission channel, and the decoding algorithm (including the [maximal] number of iterations) being fixed. The channel noise space \mathcal{N} is then divided into two sets: $\mathcal{N} \setminus \mathcal{E}$ and \mathcal{E} , noise realizations that are decoded successfully and the ones that result in the decoding error. The *instantons* are defined as the positions of local maxima of the noise distribution density $P(\xi) = \prod_{i=1}^N P(1 - \xi_i|+)$ over the set of error causing noise configurations \mathcal{E} . In the limit of high SNR the probability of the decoding error somewhere in the information block, Frame-Error Rate (FER), is controlled by the instanton with maximal $P(\xi)$ and its vicinity. (In order to describe the FER vs. SNR dependence in the moderate SNR region one may need to collect the contribution from several instantons.) Such a definition of instanton is a paraphrasing of “source of trouble”, and is practically useless without a method to locate it.

The noise configuration is said to *withstands* n iterations if after n iterations the decoding output is still wrong (that includes the case when the decoding output is a wrong but valid codeword, the noise then withstands ∞ iterations). Checking for the output being a codeword at each iteration makes the set \mathcal{E} being a non-increasing function of the number of iterations: $\mathcal{E}(n_{\text{iter}} + 1) \subseteq \mathcal{E}(n_{\text{iter}})$.

The min-sum decoding is the high SNR limit of the sum-product algorithm. In addition, if $P(1 - \xi|+) = \exp(-\beta(\text{SNR}) \cdot F(\xi))/Z(\text{SNR})$ for some increasing function $\beta(\text{SNR})$, then the decoding input has the form $h(\xi) = \beta \cdot (F(2 - \xi) - F(\xi))/2$. (This includes Additive White Gaussian Noise (AWGN) channel with $F(\xi) = \xi^2$ and $h = 2\beta(1 - \xi)$.) As the min-sum decoding is scalable (*i.e.*, the result of the decoding stays the same if the decoding input vector \mathbf{h} is multiplied by a positive number), the set \mathcal{E} is independent

This work was supported by NSF grant DMS-0807592 “Asymptotic Performance of Error Correcting Codes”.

M. Stepanov is with Department of Mathematics, University of Arizona, Tucson, AZ 85721, USA (e-mail: stepanov@math.arizona.edu).

of SNR, so are the instantons.

II. CYCLING OF ITERATIONS

One could imagine several possibilities how the iterative decoder could fail:

- R: The iterative decoding *was converging to the right solution*, but it *didn't succeed* during the allowed number of iterations.
- W: The iterative decoding *converged but to a wrong place*. After the convergence the decoding output is a *codeword*, just not the one that was sent.
- P: The iterative decoding *converged but to a wrong place*. After the convergence the decoding output is *not even a codeword*.
- C: The iterative decoding *is not going to converge* no matter how many iterations you can afford.

The situation R can be corrected by adding more iterations. It is highly possible that in situation W even maximum likelihood decoding would make an error, and the probability of such a situation [in the presence of error floor] is very small, thus the error because of possibilities P or C is much more probable.

Following the so-called Bethe free energy variational approach [11], belief propagation can be understood as a set of equations for beliefs solving a constrained minimization problem. On the other hand, a more traditional approach is to interpret belief propagation in terms of an iterative procedure — so-called belief propagation iterative algorithm [1], [12], [13]. Being identical on a tree (as then belief propagation equations are solved explicitly by iterations from leaves to the tree center) the two approaches are however distinct for a graphical problem with loops. In case of their convergence, belief propagation algorithms find a minimum of the Bethe free energy [11], [14], [15], however in a general case convergence of the standard iterative belief propagation is not guaranteed.

Experiments with the Tanner's [155,64,20] code [16] showed the following: The instanton for linear programming decoding [7], that is minimizing a certain part of the Bethe free energy and is not iterative in nature, for AWGN channel has the effective distance close to 16.4 [17], [18]. At the same time the noise configuration ξ with effective distance or weight $w(\xi) = \sum_{i=1}^N F(\xi_i) = \|\xi\|_2^2 \approx 12.45$ which withstands 410 iterations was found [19]. There is a strong indication that in the close vicinity of this noise configuration there are ones that withstand arbitrary large number of iterations.

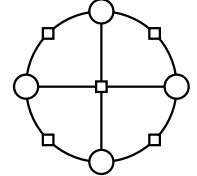
If the decoder provides errors mostly due to situation P, then it converges in most occasions. The fixed point of iterative decoding is the minimum of Bethe free energy. Thus, the iterative decoder should work not worse than linear programming decoder, as the latter neglects a certain part of Bethe free energy. That contradicts to what was observed experimentally for the Tanner's [155,64,20] code: $12.45 < 16.4$.

In contrast with the decoding algorithms which are static (e.g., linear programming decoding), in the case of iterative decoding the instantons could be inherently dynamic, and in order to find them the dynamics of iterations [in full details] should necessarily be considered.

k	$m_1^{(k)}$	$m_2^{(k)}$	$m_3^{(k)}$	$m_4^{(k)}$
0	-3	1	3	3
1	2	-2	6	2
2	4	8	-2	6
3	8	4	12	-2
4	-2	12	4	20
5	30	-2	14	4
6	4	38	-2	18
7	20	4	48	-2
8	-2	24	4	56
...
$4n$	-2	$12n$	4	$36n - 16$
$4n + 1$	$36n - 6$	-2	$12n + 2$	4
$4n + 2$	4	$36n + 2$	-2	$12n + 6$
$4n + 3$	$12n + 8$	4	$36n + 12$	-2

Fig. 1. Decoding dynamics on the instanton $\xi = (10, 6, 4, 4)/7$. The vector \mathbf{h} is proportional to $(-3, 1, 3, 3)$, and (as the decoding is scalable) the latter was used as \mathbf{h} to form the table. The general formula at the end starts to be applicable from $n \geq 1$, while at iteration $k = 2$ it is not valid yet.

As an example of cycling of iterations, consider a simple code with 4 bits and 5 parity checks:

$$\hat{H} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$


The parity checks are obviously redundant, and the code has 2 codewords: $(+, +, +, +)$ and $(-, -, -, -)$. As the first 4 parity checks have connectivity 2, the only pseudo-codewords are the codewords. Because of the checks with connectivity 2 all the bits (even on a graph-cover) should have the same values.

The lowest instanton that survives infinite number of iterations is $\xi = (10, 6, 4, 4)/7$ with the weight $w(\xi) = (10^2 + 6^2 + 4^2 + 4^2)/7^2 = 168/7^2 = 24/7 < 4$ (the numeration of bits goes along the 8-cycle containing the checks with connectivity 2).

The cycling dynamics of iterations is shown at Fig. 1. The decoding output (and the messages bits \leftrightarrow checks) is not exactly periodic with the iteration number. If one considers one iteration of the decoder as a mapping in the space of messages η , then the instantons are not necessarily periodic orbits (i.e., exact cycles) of the mapping.

III. INSTANTONS ARRAY

The instanton-amoebe scheme [9], [19] while being quite effective in getting instantons for small number of iterations n_{iter} (with about 10 iterations being the maximum in practice) is having difficulties in finding the instantons for large n_{iter} . The problem is with the rough landscape of the function amoeba tries to optimize. The moves amoeba does do assume that the landscape is regular (see [20], [21]). The problem with the application of downhill simplex/amoeba method to finding n_{iter} instantons is that amoeba always aims for noise configurations that withstand n_{iter} (i.e., many) iterations. The

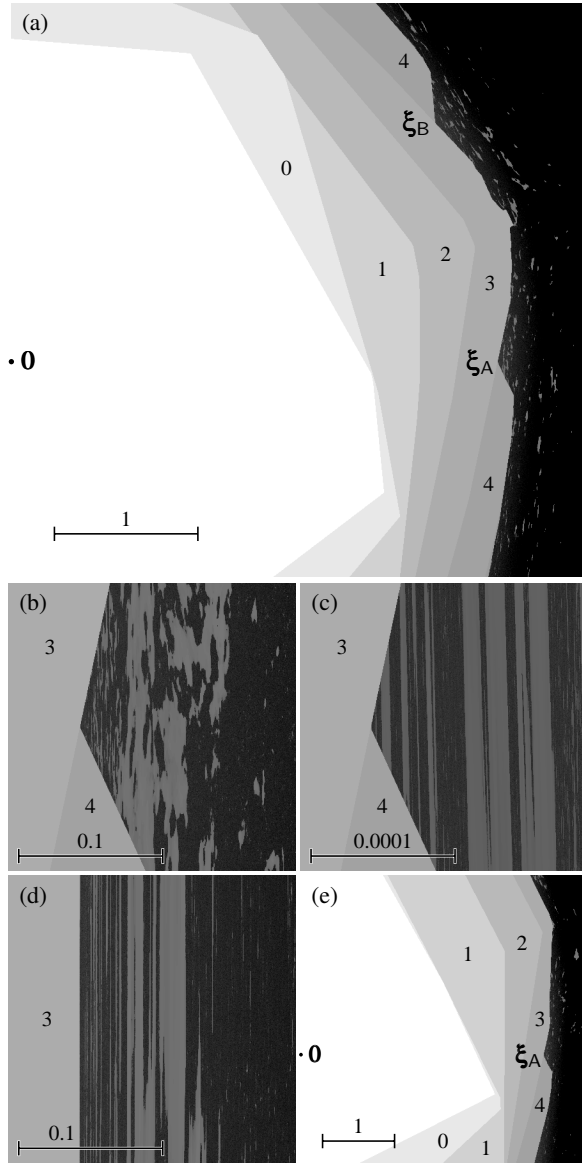


Fig. 2. Two-dimensional cuts of the [155-dimensional] noise space that contain zero noise vector $\mathbf{0}$ and the lowest weight instanton ξ_A for Tanner's [155,64,20] code and AWGN channel. The line going through $\mathbf{0}$ and ξ_A is horizontal. The plane of the cut is determined by the 3rd point it goes through. In panels (a), (b), and (c) it is the instanton ξ_B ; in panel (d) it is a random vector; and in panel (e) it is the vector \mathbf{t} with $t_{60} = t_{122} = t_{130} = t_{131} = t_{136} = 1$ and all other components being 0. The labels 0, 1, 2, 3, and 4 indicate how many iterations the noise withstands in this area. The tone of gray is calculated as $(9 - \log_2 n)/11$, where n is how many iterations the noise configuration withstands, with 0/1 being black/white. Tones 10/11 and 1 correspond to $n = 0$ and correct decoding without any iterations (i.e., $\xi_i < 1$ for all i).

set $\mathcal{E}(n_{\text{iter}})$ of such noise configurations [for large n_{iter}] is very irregular near its boundary (see Fig. 2), and amoeba is getting confused and uncontrollably reduces its size without any progress.

The algorithm shown in Fig. 3 and described below overcomes this difficulty and is able to find instantons for large n_{iter} . The procedure deals with the array of noise configurations, $\xi(k)$, $k = 0, 1, \dots, n_{\text{iter};\text{max}}$, where at any time the noise $\xi(k)$ is the one with the largest $P(\xi)$ (or the lowest weight $w(\xi) = \sum_{i=1}^N F(\xi_i)$) from all the withstanding k iterations noise

- | | |
|----|--|
| L1 | start with the noise vector $\xi = (1, 1, \dots, 1)$ |
| L2 | check some (may be empty) list of noise vectors |
| L3 | for $k = 0, 1, \dots, n_{\text{iter};\text{max}}$ |
| L4 | perturb $\xi(k)$ |
| L5 | check perturbed noise vector |
| L6 | go to L3 or exit |

Fig. 3. Iterative decoding instanton search algorithm.



Fig. 4. The two lowest instantons ξ_A and ξ_B . The tone of gray is calculated as $1 - \xi/2$, with $\xi = 2 / \xi = 0$ being black/white. The 155×93 parity check matrix \tilde{H} consists of three blocks: $(\hat{R}^1 \hat{R}^2 \hat{R}^4 \hat{R}^8 \hat{R}^{16})$, $(\hat{R}^5 \hat{R}^{10} \hat{R}^{20} \hat{R}^{18})$, $(\hat{R}^{25} \hat{R}^{19} \hat{R}^7 \hat{R}^{14} \hat{R}^{28})$, where \hat{R} is the 31×31 matrix that cyclically shifts a column vector up by one component.

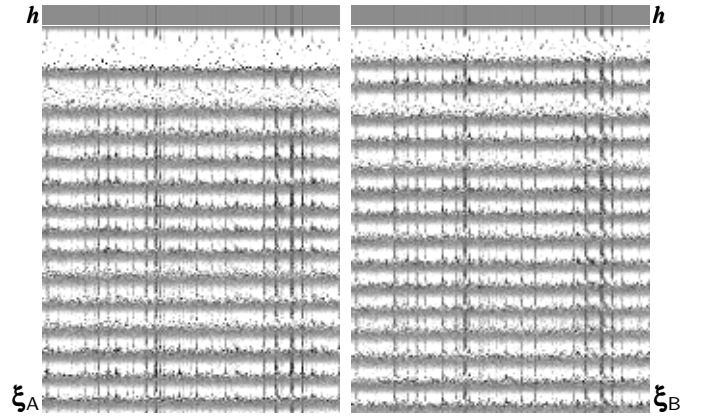


Fig. 5. Iterative decoding output m on the instantons ξ_A and ξ_B , 200 iterations running from top to bottom are shown. The tone of gray is calculated as $(1 + m/10)/2$, with 0/1 being black/white. Middle gray (tone 1/2) corresponds to undecided output $m = 0$. The decoding input $h = m^{(0)}$ is shown at the top for comparison of input and output magnitudes.

configurations that were encountered in the procedure so far. (The updates of $\xi(k)$ are done in the line L5 and (at the start) in the line L2.)

In the line L1 of the algorithm the output of the channel is completely undecided ($\mathbf{h} = (0, 0, \dots, 0)$). This configuration obviously withstands ∞ iterations, although $P(\xi)$ at it is quite low. This step makes $\xi(k) = (1, 1, \dots, 1)$ for all $k = 0, 1, \dots, n_{\text{iter};\text{max}}$.

In the line L2 the noise configurations that are known from some external source (e.g., from previous runs of the procedure or from the analysis of trapping sets or pseudo-codewords) may be introduced as a starting point for instanton search.

This procedure, applied to Tanner's [155,64,20] code and AWGN channel, with $n_{\text{iter};\text{max}} = 100$, produced an instanton ξ_A with the lowest weight $w(\xi_A) = \|\xi_A\|_2^2 < 11.475333$ that causes iterations to cycle with the period of length 12 (see Fig. 5). The next instanton ξ_B has the weight $w(\xi_B) \approx 11.4996$. The differences in weight for configurations that withstand 20 or more iterations are very small. Submitting the array $\xi(k)$ as an initial state of the procedure with larger $n_{\text{iter};\text{max}}$ relatively quickly produces noise configurations with very close weight that withstand larger $n_{\text{iter};\text{max}}$ iterations.

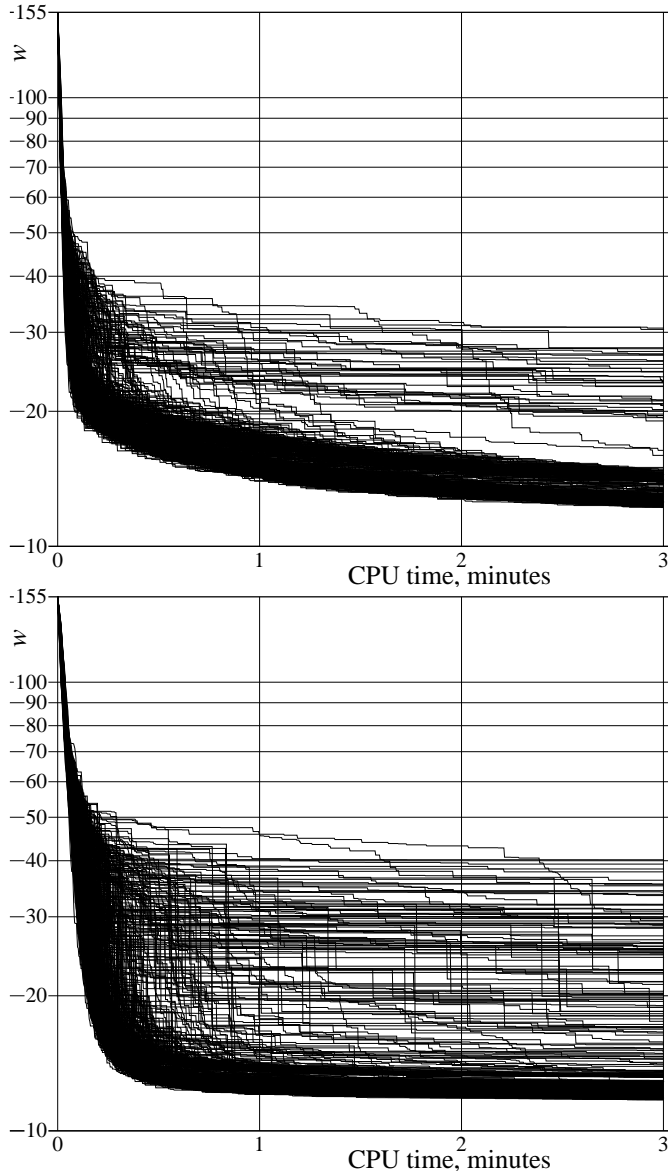


Fig. 6. The effective weight w of the withstanding $n_{\text{iter},\text{max}} = 100$ iterations noise configuration for the Tanner's [155, 64, 20] code and AWGN channel vs. CPU (Intel Xeon X3360, 2.83 GHz) time, 500 realizations are shown. The feedback for the amplitude a is A (upper panel) and D (lower panel).

Below are the details of the procedure used to generate Figs. 6 and 7¹. The noise vector ξ is perturbed as $\xi \rightarrow c\xi + a\psi$, where the components of ψ are independent standard normal random variables. The coefficient $c = \sqrt{1 - a^2 N / w(\xi)} < 1$ makes the expected value $E\|c\xi + a\psi\|_2^2 = c^2 w(\xi) + a^2 N = w(\xi)$ not being systematically increased by the addition of $a\psi$.

One doesn't want to have the amplitude of the perturbation a being too small (or the optimization is slow) or too large (then the perturbed noise is rejected often). To accelerate the procedure the amplitude a is chosen according to the following negative feedback: Each noise configuration ξ has a number A attached to it, and the perturbed noise $c\xi + a\psi$ gets the number $2A$ attached, while the number attached to ξ is decreased by

¹The perturbation of noise vector in the line L4 (including the choice of the perturbation amplitude), of course, can be done in many different ways.

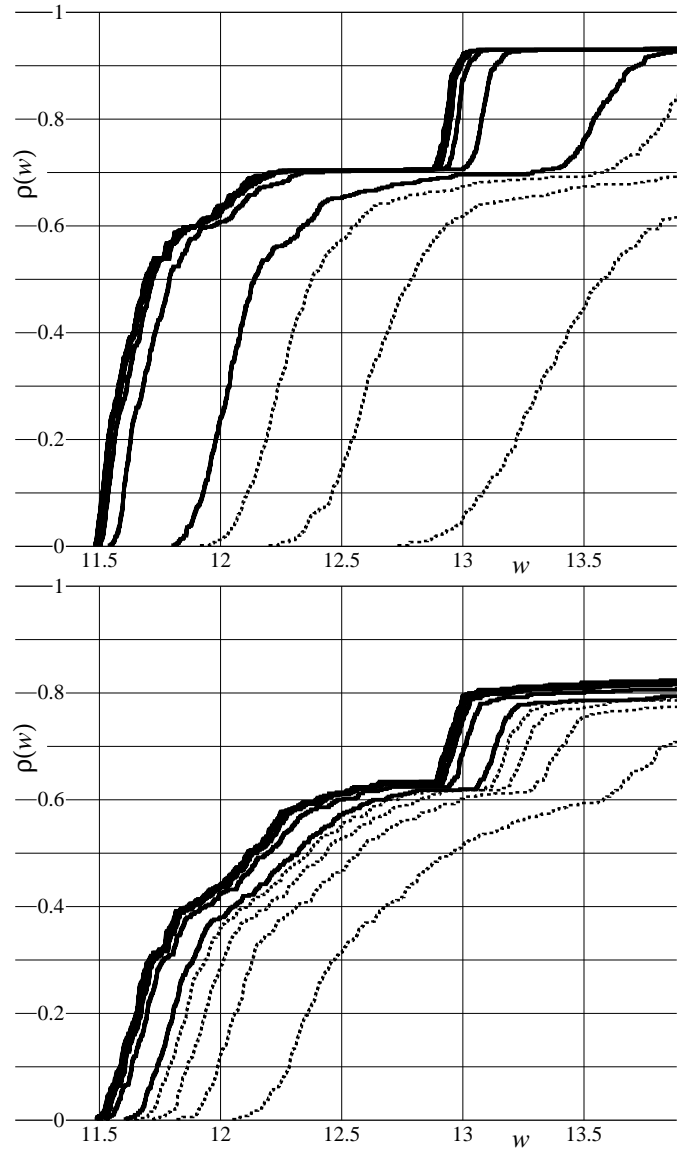


Fig. 7. The probability/frequency of occurrence, $p(w)$, of the withstanding $n_{\text{iter},\text{max}} = 100$ iterations noise configurations with the weight w or smaller after 1, 2, 3, 4 (dashed curves) and 5, 10, ..., 30 (solid curves) minutes of CPU time. Code, channel, feedback, and CPU are the same as in Fig. 6.

a factor 0.999. In the line L1 the value $A = 0.1$ is attached. The amplitude of the perturbation a is chosen as A: $a = A$; D: $0.1A < a < A$; and W: $10^{-14} < a < 0.1$, with uniform distribution of $\log a$ in both D and W. In comparison to A and D, the progress in W is slow — the perturbation amplitude a is often too small or too large.

How $w(\xi(100))$ goes down with time is shown in Fig. 6. It can be seen that sometimes $w(\xi(100))$ suddenly drops down quite a bit — it is happening when beginning part of the array (but not $\xi(100)$) already went lower in weight, and then suddenly a small perturbation withstands 100 iterations, so $\xi(100)$ is updated. Such events are what makes the whole procedure work. The progress in the beginning part of the array is a lot more regular, and the procedure treasures it in hope that it will be converted into the progress at $n_{\text{iter},\text{max}}$.

The distribution of $w(\xi(100))$ is shown in Fig. 7. As it can

be seen, the fate of the run is determined quite early.

At the very beginning there are few rejections of the perturbed noise vectors, and with the feedback A [with larger choices of a] the weight goes down faster than with D (see Fig. 6). When the weight reaches about 20, the feedback D is more effective, probably because eventual smaller than A choices of a lead to the perturbations being not rejected more often, which keeps the values of A large enough. Eventually A is more effective (see Fig. 7), although such a difference between A and D is a bit surprising.

IV. DISCUSSION

In the vicinity of the instantons ξ_A and ξ_B there are noise configurations that withstand just 3 iterations (see Fig. 2). Is it possible to locate these instantons from the analysis of the decoding with small number iterations?

The lowest instanton weight w describes how fast the decoding error probability goes down with SNR in the high SNR limit: $\log P(\mathcal{E}) \sim -\beta(\text{SNR}) \cdot w$. The value of w quickly saturates with the number of iterations n_{iter} , and the decrease of $P(\mathcal{E})$ with n_{iter} is probably caused by the thinning of the set $\mathcal{E}(n_{\text{iter}})$ in the vicinity of the instanton. How exactly does this happen?

In the example from Sec. II the magnitude of iterative decoder messages was growing linearly with the iteration number. Such a linear growth was not observed for the instantons ξ_A and ξ_B . Should that be expected?

REFERENCES

- [1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.
- [2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, p. 1645, 1996.
- [3] N. Wiberg, "Codes and decoding on general graphs," Ph.D. thesis, Univ. Linköping, Sweden, 1996.
- [4] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003.
- [5] T. J. Richardson, "Error floors of LDPC codes," in *41st Allerton Conf. Commun., Control and Computing*, (Monticello, IL, USA, October 1–3, 2003), pp. 1426–1435.
- [6] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, 2002.
- [7] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, 2005.
- [8] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. J. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. Commun.*, vol. 57, no. 11, pp. 3258–3268, 2009.
- [9] M. G. Stepanov, V. Chernyak, M. Chertkov, and B. Vasic, "Diagnosis of weaknesses in modern error correction codes: a physics approach," *Phys. Rev. Lett.*, vol. 95, no. 22, p. 228701, 2005.
- [10] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, 1981.
- [11] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Trans. Inf. Theory*, vol. 51, no. 7, pp. 2282–2312, 2005.
- [12] J. Pearl, *Probabilistic reasoning in intelligent systems: network of plausible inference*. San Francisco: Kaufmann, 1988.
- [13] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, 1999.
- [14] S. C. Tatikonda and M. I. Jordan, "Loopy belief propagation and Gibbs measures," in *18th Conf. Uncertainty in Artificial Intelligence*, (Edmonton, AB, Canada, August 1–4, 2002), pp. 493–500.
- [15] T. Heskes, "On the uniqueness of loopy belief propagation fixed points," *Neural Computation*, vol. 16, no. 11, pp. 2379–2413, 2004.
- [16] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *6th Intl. Symp. Commun. Theory Appl.*, (Ambleside, UK, July 15–20, 2001).
- [17] R. Koetter and P. O. Vontobel, "Graph covers and iterative decoding of finite-length codes," in *3rd Intl. Conf. Turbo Codes and Related Topics*, (Brest, France, September 1–5, 2003), pp. 75–82.
- [18] M. Chertkov and M. G. Stepanov, "An efficient pseudocodeword search algorithm for linear programming decoding of LDPC codes," *IEEE Trans. Inf. Theory*, vol. 54, no. 4, pp. 1514–1520, 2008.
- [19] M. G. Stepanov and M. Chertkov, "Instanton analysis of low-density parity-check codes in the error-floor regime," in *2006 IEEE Intl. Symp. Inf. Theory*, (Seattle, WA, USA, July 9–14, 2006), pp. 552–556.
- [20] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.
- [21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C: the art of scientific computing*. Cambridge: Cambridge University Press, 1988.